计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20148038
52(11):2568-2576,2015

网络分簇 BWRAID:更快的扩展、恢复和读写性能

孙振元[1,2,3] 许 鲁[1,2,3] 刘振军[1,2,3] 董欢庆[1,2] 刘 昌[1,2]
[1](中国科学院计算技术研究所 北京 100190)
[2](中科蓝鲸信息技术有限公司 北京 100089)
[3](中国科学院大学 北京 100049)
(sunzhenyuan@nrchpc.ac.cn)

Network Declustering BWRAID: Faster Scalability, Recovery and IO Performance

Sun Zhenyuan[1,2,3], Xu Lu[1,2,3], Liu Zhenjun[1,2,3], Dong Huanqing[1,2], and Liu Chang[1,2]
[1](*Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190)
[2](*Zhongke Blue Whale Information Technologies Co., Ltd, Beijing* 100089)
[3](*University of Chinese Academy of Sciences, Beijing* 100049)

Abstract Storage area network (SAN) is important for network storage. We construct BWRAID, a distributed RAID in a SAN, from commodity hardware. The original version BWRAID has a symmetric architecture. However, it has three problems: Firstly, it reads data to re-calculate parity when it expands, and the re-calculation consumes much IO and time. Secondly, it recovers data to one storage node (SN), and recovery can be more efficient if parallel on multi nodes. Thirdly, its data layout is bad for IO, that makes its internal RAID4 have many costly read-modify-write updates even on sequential writes. To solve these problems, we propose a network declustering BWRAID. It has an asymmetric architecture like a declustering RAID, but it is declustered by equal size virtual disks instead of blocks. It is expanded by moving virtual disks without calculation. It runs multi recovery in parallel as the number of nodes involved in each recovery is less than the total number of nodes in the system. To optimize IO, we change its data layout to express user IO space by internal RAID4 stripes. We also provide algorithms to search suitable virtual disks for system allocation, expansion, or recovery. Experiments show that network declustering BWRAID is better than the original one. It expands the system without calculating parity five to eight times faster, and its parallel recovery is multi times faster, and it increases the IO performance with the new data layout.

Key words network RAID; declustering RAID; data layout; scalability; recovery; virtual storage

摘 要 存储区域网(storage area network, SAN)是重要的网络存储方法.使用商用硬件 BWRAID 在 SAN 上实现了分布式 RAID.初始版本的 BWRAID 使用全对称结构,然而其存在 3 个问题:1)扩展时要读取数据重新计算校验,IO 负载高、扩展时间长;2)将数据集中恢复到单个存储节点,没有分布的并发恢复;3)数据布局不合理,导致内部 RAID4 有大量同步更新.为解决上述问题,提出了“网络分簇 BWRAID”.新系统采用“分簇 RAID”(declustering RAID)的非对称结构,分簇对象是相等大小的小虚拟盘而不是数据块;在扩展时,它在节点之间仅迁移虚拟卷,不需计算校验.由于一个恢复需要的节点数

收稿日期:2014-09-19;**修回日期**:2015-01-14
基金项目:国家“八六三”高技术研究发展计划基金项目(2013AA013205);国家“九七三”重点基础研究发展计划基金项目(2011CB302304);国家科技支撑计划基金项目(2011BAH04B04);中国科学院战略性先导科技专项基金项目(XDA06010401);中国科学院重点部署基金项目(KGZD-EW-103-5(7))

量小于节点总数,多个恢复就能并行.为优化 IO 使用新的数据布局,按内部 RAID4 条带组织用户的存储空间,并给出了搜索虚拟盘的算法,用于在系统分配、扩展、恢复时,搜索合适的虚拟盘.实验表明网络分簇 BWRAID 更好:在系统扩展时无需重新计算校验,加速扩展 5～8 倍;并行恢复成倍加速;新数据布局提高了 IO 性能.

关键词 网络 RAID;分簇 RAID;数据布局;可扩展性;恢复;虚拟存储

中图法分类号 TP302

对于需要高速"块级别"访问的服务器,存储区域网(storage area network, SAN)仍然是重要且常用的方法之一,比如电子邮件服务器、数据库、高利用率的文件服务器等.RAID[1]将多个硬盘组成阵列,是提供高性能可靠存储的流行方案.BWRAID 在通用硬件上实现高可用可扩展的 SAN 网络 RAID.本文称第一个 BWRAID 系统[2-3]为"初始 BWRAID",它在存储服务器、也称为存储节点(storage node, SN)使用全对称的冗余结构,拥有 RAID1 的 IO 性能和 RAID4 的存储效率;同时,也具有 RAID1 更新双份数据的网络开销,RAID4 恢复数据(重构)的高开销.这种转换策略也存在于 AutoRAID[4],Disk-Reduce[5]等系统中.显然,恢复时间越短,BWRAID 的实用价值越高.网络存储环境下,为了增加系统的吞吐率和存储容量,BWRAID 的可扩展性很重要.

然而,初始 BWRAID 的全对称结构导致了 SN 间的紧耦合,存在 3 个问题:1)扩展效率低.初始 BWRAID 以 SN 为单位进行扩展,负载高且耗时.虽然先前的研究[2]提出了优化扩展的方法,但是仍然需要计算校验、扩展耗时.2)恢复效率低.恢复的数据集中写入单个 SN,没有发挥多 SN 并行处理的能力.3)它的数据布局对 IO 性能有负面影响,导致它的内部 RAID4 存在大量同步更新,这抵消了 BWRAID 的异步计算校验的优势.

为了应对 SN 的数量变化和容量变化,高效扩展的关键在于:复用原有的校验,即维持原有 RAID 的组成和大小,使 RAID 条带宽度与 SN 的数量无关.用于加快数据恢复速度的"分簇 RAID"(declustering RAID)[6]技术满足该需求.在一般 RAID 中,条带宽度等于硬盘数量,恢复单块数据需要所有硬盘参与.而在分簇 RAID 中,条带宽度小于硬盘数量,恢复单块数据只需要部分硬盘参与,其他的硬盘可以并行恢复其他数据.分簇 RAID 核心优点是 RAID 条带宽度无需等于存储设备数量,这样在扩展时可不改变 RAID 组成,复用校验.同时,相关研究表明:在使用相同宽度时,分簇 RAID 和一般 RAID 具有一样的数据可靠性[7].

本文将分簇 RAID 技术推广到网络 RAID,主要贡献是:1)提出并实现了以小容量"逻辑卷"(logic volume, LV)为单位构造网络分簇 RAID 的方法,而不是使用"数据块"(chunk)构造;2)提出并论证了网络分簇 RAID 存储空间管理的算法,包括分配、扩展、恢复时搜索逻辑卷的算法;3)改进了 BWRAID 结构和其客户端的组成方法,使用合理的数据布局提高了 IO 性能.

1 初始 BWRAID

1.1 系统结构

初始 BWRAID 采用以数据块为单位的全对称结构,如图 1 所示,该结构直接从 RAID1 演进.多个 SN 形成冗余组,条带宽度等于 SN 的数量.按它的映射公式[3],逐步计算出每个校验块并按循环方式(round robin)分布.设 $P(X,Y)$ 表示数据块 X 与数据块 Y 的校验块.

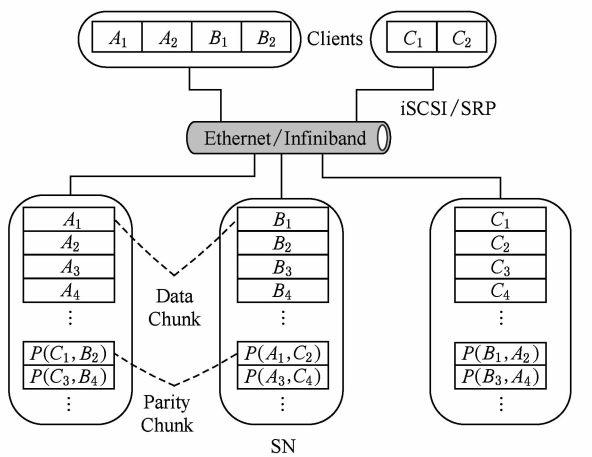


Fig. 1 Architecture of original BWRAID.

图 1 初始 BWRAID 系统结构

BWRAID 系统架构采用计算和存储分离的设计[8-9].应用服务器提供计算服务,SN 集群提供存储服务.块设备软件堆栈提供了核心的存储功能,包括:

逻辑卷、快照、RAM 缓存、硬盘介质缓存(disk as cache, DCache)、RAID1 和 RAID4. 逻辑卷设备按日志方式管理存储池,使用类似日志结构文件系统(log structured file system,LFS)[10]的追加写技术,对外提供逻辑卷,逻辑卷也称为虚拟盘(virtual disk);快照建立在日志结构的基础上,用于隔离数据更新;RAM 缓存为日志提供固定大小的块,应用 P-Cache[11]技术,多个缓存可以动态共享缓存池;DCache 作为大容量写缓存,尽可能形成连续的数据块,以减少写入下层 RAID4 的开销.

1.2 DPP 布局的设备堆栈

为使各个 SN 在数据块上"全对称",初始 BWRAID 中多个 SN 交错成为全对称结构,如图 2 所示. 虚线表示网络连接,箭头表示网络导出方向,实线表示本地连接. "条带化"(stripe)将一个存储空间循环映射到多个存储设备,而"逆条带化"(de-stripe)是"条带化"的逆映射.

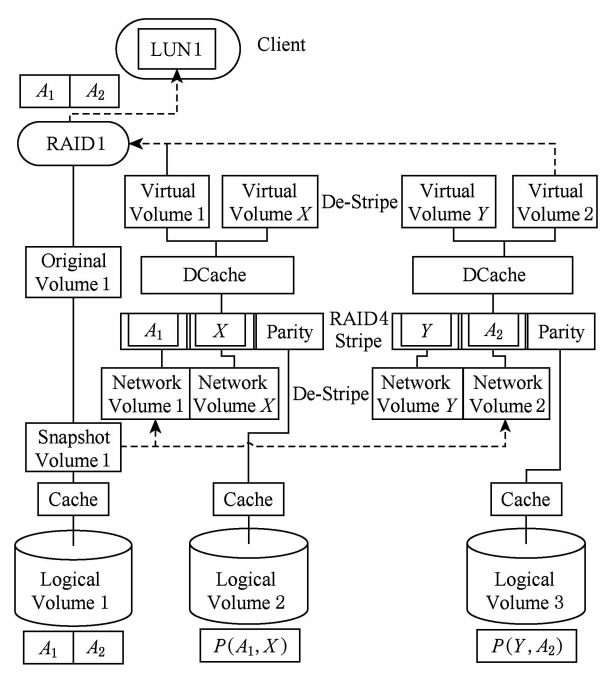


Fig. 2 Device stacks of DPP data layout.

图 2 DPP 布局的设备堆栈

因为数据聚集在一个 RAID1 上,离散到多个 RAID4 上参与多个校验计算,本文称这种数据布局为 DPP(data, parity, parity). DPP 布局的问题是:一个应用的数据写到多个 SN 上的缓存中,一个缓存中混合了来自多个应用的数据,已经失去应用的写请求访问模式. 因而,缓存很难形成连续数据,使下方 RAID4 上以同步更新为主,增加了系统的负载. 可见,DDP 布局对 BWRAID 的性能产生了负面影响.

2 网络分簇 BWRAID 设计

2.1 系统结构

网络分簇 BWRAID 按相同大小的 LV 为单位在网络间分簇,是非对称结构,如图 3 所示. 将 SN 的空间分为很多容量相等的小 LV,小 LV 作为数据卷或校验卷. 小 LV 通过网络组合成 RAID 卷,RAID 卷的条带宽度小于等于 SN 的数量,许多 RAID 卷散列在不同 SN 上.

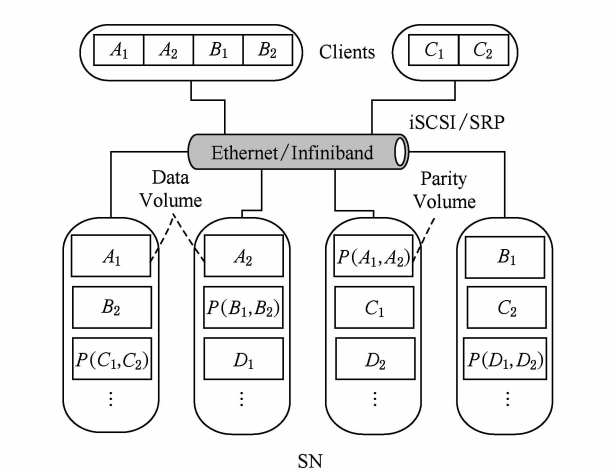


Fig. 3 Architecture of network declustering BWRAID.

图 3 网络分簇 BWRAID 系统结构

2.2 DDP 布局的设备堆栈

为了优化性能,本文提出了新的数据布局,称为

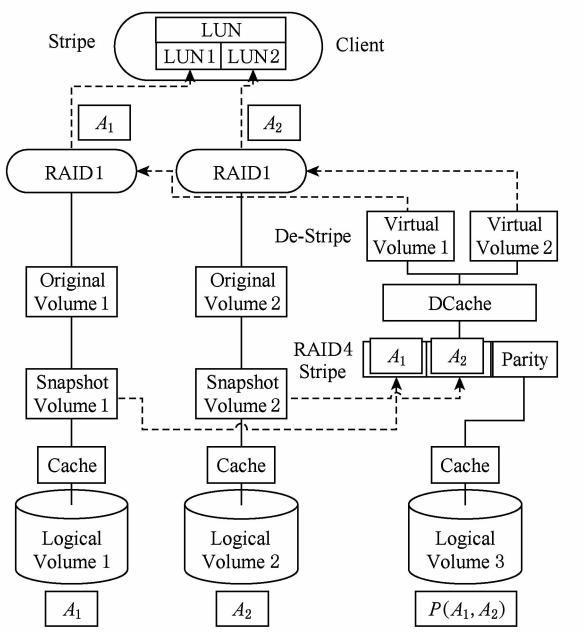


Fig. 4 Device stacks of DDP data layout.

图 4 DDP 布局的设备堆栈

DDP(data,data,parity). 数据离散在多个 RAID1 上,聚集到一个 RAID4 上,如图 4 所示. 客户端按 RAID4 条带映射关系来组织存储空间,这样客户端的地址空间可按 RAID4 条带大小划分.

DDP 是 DPP 的等价异构体. DPP 布局经"逆条带化"导出给客户端后重新"条带化"可转换为 DDP 布局,但是转换后在 RAID1 内部混合了不同应用的访问. 与之相比,DDP 布局在 RAID1 和 RAID4 上都没有混合不同应用的数据,并且结构更为简洁清晰.

2.3 网络分簇规则

为加快在线扩展和数据恢复速度,需要将分簇 RAID 推广到网络 RAID. 本文提出了如下网络分簇 RAID 的构造规则:1) 要求所有 LV 容量相等;LV 有全局唯一的名字标识,系统通过名字查找 LV;LV 可在 SN 之间迁移,不需映射公式. 2)由多个 LV 通过网络组成"RAID 卷";自 RAID 卷创建以后,其 LV 成员维持不变,直到删除;RAID 卷宽度小于等于可用的 SN 数量. 3)网络分簇 RAID 由许多 RAID 卷组成;以 RAID 卷为单位分配资源给客户端使用;客户端可以组合使用多个 RAID 卷.

本文将按上述规则用容量相等的小 LV 组成 DDP 布局的设备堆栈称为 BWRAIDlet,将所有 BWRAIDlet 和客户端构成的集群系统称为"网络分簇 BWRAID".

虽然 BWRAIDlet 不对称,但是可通过合理分布 LV,实现粗粒度的容量均衡. 在分配、扩展、恢复时,核心问题是找到要处理的 LV,不需要映射公式或维持对称的结构. 例如,扩展可规约为搜索 LV 的位置问题,首先在 SN 找到可流出的 LV,然后找到可流入某个 SN 的 LV. 类似地,在数据恢复时,核心问题是要搜索到可以并行恢复的 LV.

3 网络分簇相关算法

3.1 核心数据结构

本文算法的核心数据结构是二维双向链表. 二维链表中的基本对象表示 1 个 LV,包括它的容量信息和 IO 负载信息. 每个 LV 具有 2 个双向链表结构,分别用于水平链表和垂直链表. 水平链表的头部属于 1 个 RAID 卷对象(RAID volume, RV),包括这个 RV 的 IO 负载信息;垂直链表的头部属于 1 个存储节点对象 SN,包括它的已分配 LV 数量、存储容量等信息. SN 连入按它的 LV 数量排序的双向链表 SN_List. 为便于通过 LV 找到它所在的 SN 或 RV,LV 数据结构中具有二者的指针.

3.2 随机分配算法

本地分簇 RAID 使用映射方法 BIBD(balanced incomplete block designs)[6] 来实现校验块均衡分布在所有硬盘上. 然而,当硬盘数量扩展时,这种布局方法并不适用. 相关研究中没有实现或描述如何从一个 BIBD 分布变换为另一个 BIBD 分布. 另外,BIBD 中所有条带的宽度一样,而本文算法允许 RAID 卷具有不同的条带宽度.

本文采用随机算法分布 RV. 因为随机分配可以灵活地重新布局,在扩展或恢复也不需要维持原有的分布规律,而且能够获得近似平衡的分布。为提供 SN 为单位的冗余保护,规定 RV 的分布限制:RV 中任意 2 个 LV 不在同一个 SN 上.

本文采用的随机算法为:首先,从所有 SN 中随机选择 w 个节点,w 等于 RV 的宽度;然后,从 w 个节点随机选择 1 个作为校验节点,剩余的节点作为数据节点. 令 n 为 SN 总数,容易获得算法复杂度为 $O(n^2)$的随机算法.

3.3 简单的扩展算法

扩展算法是 1 种均衡算法,目标为使各个 SN 上 LV 数量均衡且 LV 的迁移次数最少. 假设在 SN 的有序链表 SN_List 上,链表头的 SN 上 LV 数量最多,链表尾的 SN 上 LV 数量最少. 迁移方法:从 SN_List 链表头的 SN 中任意选择 1 个 LV,迁移到链表尾的 SN 中,并按 LV 数量用冒泡算法重新排序这 2 个 SN. 迁移条件:2 个 SN 的 LV 数量差大于等于 2. 终止条件:SN_List 的链表头 SN 和链表尾 SN 中 LV 的数量差小于等于 1,则终止状态为 x,…,x,$x-1$,…,$x-1$. 其中 x 为 1 个 SN 上的 LV 数量. 可证明上述算法 LV 的迁移次数最少.

证明. LV 迁移次数最少,只需证明上述算法不存在循环迁移. 令 p 为任一 SN,$Out(p)$表示节点 p 流出 LV. 假设 $\exists p: Out(p)$,令 SNs 为除 p 外其他 SN 的集合,$f(k)$表示节点 k 的 LV 数目,$F(k)$表示其他节点 LV 数量最大值,有 $F(k)=\max\{f(m)\mid m\in SNs\}$. 令 $f(p)=y$,当 p 最后一次流出,有 $f(p)\geqslant F(p)\rightarrow \forall q\in SNs: f(q)\leqslant y$. 此后,$p$ 的 LV 数量 $f(p)=y-1$. 当 p 第一次流入 LV 时,由迁移条件有 $\exists q\in SNs: f(q)\geqslant y-1+2=y+1$,这与前面 $\forall q\in SNs: f(q)\leqslant y$ 矛盾,即不存在先流出后流入的节点 p. 同理可证,不存在先流入后流出的节点 p. 因此,上述算法不存在循环迁移,LV 迁移次数最少. 证毕.

简单的扩展算法移动 SN 次数很多,没有考虑 RV 分布上的限制,也没有兼顾 IO 负载均衡.下面本文描述改进的算法,满足 RV 分布限制条件,不需要移动 SN 对象,同时维持移动最少 LV 的优点.

3.4 改进的扩展算法

假设当前 LV 总数量为 S,扩展后总 SN 数量为 n,终止状态 $x,\cdots,x,x-1,\cdots,x-1$.假设有 i 个节点 LV 数量为 x,那么有 $n-i$ 个节点的 LV 数量为 $x-1$,LV 总数 $S=x\times i+(x-1)\times(n-i)$,则 x, i 为

$$\begin{aligned} x &= (S+n)/n, \\ i &= (S+n) \bmod n. \end{aligned} \tag{1}$$

改进的扩展算法概要流程如下:首先,根据终止状态的 x,i 计算出每个 SN 需要流出的 LV 数量;然后,依据 RV 的分布限制并参考负载比例,搜索每个 SN 上可流出的 LV;最后,依据 RV 分布限制,同时兼顾负载均衡,标记这些 LV 的目的 SN.

上述算法关键是:在 SN 上能够搜索到符合 RV 分布限制的 LV,且使 SN 达到终止状态.令 C 表示流入 LV 的 SN 数量,r 表示任一 RV,l 表示任一 LV,$S(r)$表示 r 上已经流出的 LV 的集合,$R(l)$表示 l 所在的 RV.为保证 RV 的分布限制,如果要流出 r 上的 LV,则必须满足条件 $|S(r)|<C$.令 $L(p)$ 表示节点 p 上的 LV 集合,则当 p 达到终止状态时其 LV 的数量 $|L(p)|=x$ 或 $|L(p)|=x-1$.可证明,对未达到终止状态流出 LV 的节点 p,有如下命题 $|L(p)|=g+x-1\rightarrow|M|\geqslant g$ 成立.其中 $g\geqslant 1$,M 表示节点 p 上的 LV 所属 RV 流出数量小于 C 的 LV 集合,即 $M=\{l\mid \exists l\in L(p):|S(R(l))|<C\}$.

证明. 可证明 $|M|<g$ 不成立.令 O 表示目前流出 LV 的总数量,I 表示目前流入 LV 的总数量,$I=O$.由于存在未达到终止状态的需要流出的节点 p,根据终止状态,有 $I<x\times C$.算法中 RV 流出数量为 C 后不被选择,$L(p)-M$ 表示节点 p 上 LV 所属 RV 流出数量等于 C 的 LV 集合.若 $|M|<g$,$|L(p)|=g+x-1$,则 $|L(p)-M|>g+x-1-g=x-1$,即 $|L(p)-M|\geqslant x$.而 $O\geqslant|L(p)-M|\times C\geqslant x\times C$,与前面 $O=I<x\times C$ 矛盾,因此 $|M|<g$ 不成立.

证毕.

搜索 LV 时可考虑负载均衡.计算流出 LV 的 1 个 SN 期望流出负载值的方法如下:依据 1 个 SN 占总负载的比例,以及流入 LV 数量占它 LV 总数的比例,求出这个 SN 期望流出的负载值.计算流入 LV 的 1 个 SN 期望流入负载值的方法如下:当前已经获得需要迁移的所有 LV,计算它们的总负载;然后依据流入 LV 的 SN 的负载值,求出这个 SN 期望流入的负载值.在为 1 个 SN 搜索 LV 时,尽量满足这些期望的负载值.

改进扩展算法的时间复杂度为 $O(n\times v\times g)$.其中,n 为扩展前 SN 的数量;v 为扩展前 SN 上 LV 数量均值;g 为 SN 扩展时的 LV 数量差值,即搜索的 LV 数量;在 1 个 SN 上搜索 g 个 LV 的复杂度为 $O(v\times g)$.设 a 为新增加的 SN 数量,根据 LV 总数有 $g=a\times v/(n+a)<a\times v/n$,则扩展算法复杂度小于 $O(a\times v^2)$,即搜索时间可由新增节点数量和 LV 均值界定.

获得需要迁移的 LV,并标记它们的来源 SN 和目的 SN 后,即可启用在线 LV 迁移.当一次增加多个 SN 时,可进行并行数据迁移.搜索并行迁移 LV 的算法类似下一节的并行恢复算法,本文不再详述.

3.5 并行恢复算法

网络分簇 RAID 在恢复时应进行多个 SN 并行处理.为了避免多个并行恢复的 RV 所在的 SN 的集合存在交集,引起恢复进程竞争资源,本文限制存在 SN 交集的 RV 不能并行恢复,即 1 个 SN 在 1 个时间段内只能参与恢复 1 个 RV.

本文采用如下的并行恢复算法:1)主进程根据故障 SN 上的 LV 获得需要恢复的 RV 链表;2)顺序扫描该 RV 链表,查找可以并行恢复的多个 RV,启动并行恢复进程;主进程等待某个 RV 恢复完成,重新循环(顺序查找可以并行恢复的多个 RV),直到没有 RV 需要恢复.假设算法将正在使用的 SN 存放在 Hash 中,则能够并行恢复的 RV 需要满足 2 个条件:1)剩余 SN 的数量大于等于该 RV 的宽度 w;2)RV 需要使用的 SN 不在 Hash 中.SN 总数 n 和 RV 宽度 w 决定了最大可能的并行恢复数量.

本算法顺序查找可并行恢复的多个 RV,而不是全局找具有最大并行度的多个 RV,这并非最优算法,但简单迅速,复杂度为 $O(v\times w/n)$.其中,n 为 SN 数量,v 为需要恢复的 RV 数量,w 为 RV 的平均宽度.

4 实验结果与分析

4.1 测试环境

本文测试网络分簇 BWRAID 相对于初始 BWRAID 的存储性能,采用 InfiniBand[12] 网络来避免网络瓶颈.为避免混合 IO 对测试结果的影响,每个服务器

为 DCache 和 RAID4 校验使用不同的存储池,各分配一块磁盘.

测试机器包括 12 台 Dell R710 服务器和 1 台 Mellanox InfiniBand 交换机. R710 的硬件配置如下:2 个 Intel Xeon E5620 CPU;12 GB DDR3 RAM;4 个希捷 1 TB SATA Ⅱ磁盘,型号为 ST3100528AS;1 个 Mellanox 10 Gbps InfiniBand HCA 卡,型号为 InfiniHost Ⅲ Lx MT25204. InfiniBand 交换机配置为:24 个端口,总交互带宽为 480 Gbps,型号为 InfiniScale Ⅲ MT47396.

为方便比较,初始 BWRAID 和网络分簇 BWRAID 的条带宽度统一设定为 3. 网络协议软件堆栈使用 iSCSI-SCST[13]. IO 测试软件使用 FIO,IO 块大小 2 MB,顺序 direct 读写. BWRAID 中数据块大小设定为 1 MB,LV 大小为 32 GB,DCache 大小为 8 GB,RAM 缓存大小为 128 MB.

4.2 算法运行时间

本文使用单元测试的方法测量算法运行时间. 设定 1 个 SN 包括 16 块 1TB 硬盘,LV 大小为 32 GB,则 1 个 SN 有 500 个 LV. 设置 RV 宽度为 3,所有 LV 的负载为 1. 其中,SN 数量分别为 10,100,1 000. 使用这些参数时,LV 的二维双向链表占用内存空间随 SN 数量线性增加.

本文算法运行结果如表 1 所示. 随机分配算法运行很快,时间开销主要是访问主存开销;扩展测试均增加 10 个 SN,在 SN 基数从 10 增加到 1 000 时,扩展算法搜索 LV 的时间没有显著增加,验证了本文的扩展算法跟 SN 的增量 a 和 SN 上的 LV 数量 v 有关;通过使恢复函数立刻返回成功的方法,获得 1 个 SN 故障时并行恢复算法本身的运行时间. 在 SN 基数从 10 增加到 1 000 时,虽然运行时间跟 SN 总数成反比,然而总恢复时间变化略有增加,因为恢复进程调度次数增加.

Table 1 Running Time and Memory Consumption of Algorithms

表 1 算法的运行时间和内存消耗

SN Count	Running Time of Algorithms/s			RAM Size/MB
	Allocation	Expand 10	Recovery	
10	0.001	0.50	0.023	0.7
100	0.001	0.51	0.024	7.6
1 000	0.012	0.63	0.026	77.2

4.3 测试不同布局的 IO 性能

为了说明 DDP 布局对性能优化的效果,本文对 2 种数据布局进行对比测试,其中 RV 均匀分布. 图 5 显示了 2 种布局的 BWRAID 的顺序写性能的比较. 在没有 DCache 回刷(write back, WB)时,写性能没有实质差异. 因为此时服务模式为 RAID1,2 种数据布局的 IO 访问模式一样. 然而,在 DCache 开始回刷数据后,服务模式接近 RAID4,DPP 布局将应用数据分散写入到多个 RAID4,在每个 RAID4 上造成大量小写更新. 而 DDP 布局 BWRAID 经过 DCache 后能形成连续的顺序写,RAID4 以全条带更新为主,DDP 布局的 BWRAID 的写带宽优势明显,比 DPP 布局提高了 170%. 2 种布局下的 BWRAID 的写带宽都在 DCache 回刷后比回刷前大幅降低. 因为使用磁盘作为写缓存的介质,回刷数据需要读取磁盘,同时顺序写入同一磁盘,磁盘同时服务 2 个数据流,所以回刷时的写带宽有显著降低. 另外,回刷触发了 BWRAID 的数据去冗,消耗了数据节点的 IO 带宽.

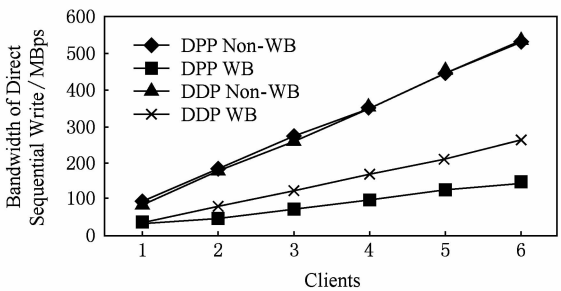


Fig. 5 Bandwidth comparison of direct sequential write.

图 5 顺序 direct 写带宽比较

图 6 对比了 2 种布局的顺序读带宽. 没有故障时,BWRAID 只从 RAID1 的本地路径读取数据. DDP 布局的 1 个客户端同时读取 2 个 SN,3 个客户端刚好利用到 6 个 SN 的读带宽. 因此,在 1～3 个客户端并行读取时,读带宽线性增加;而 4～6 个客户端并行读取时,每个 SN 上有 1～2 个访问流. 当 1 个磁盘同时服务 2 个访问地址不相关的数据流时

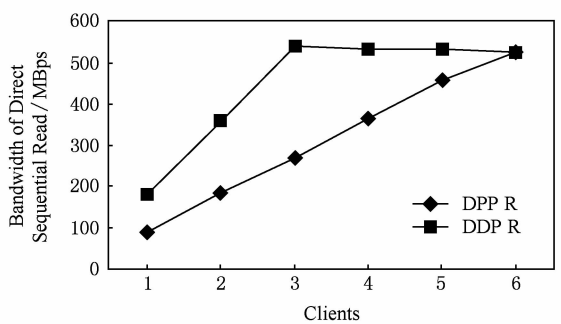


Fig. 6 Bandwidth comparison of direct sequential read.

图 6 顺序 direct 读带宽比较

引起磁头抖动,IO 性能大幅下降. 这里 RAM 缓存预读服务这种多流并发访问,使磁盘上仅有来自缓存的单流访问. 因此,4～6 个客户端并行读取时,DDP 布局的 BWRAID 仍然有较好的读性能.

4.4 测试网络分簇性能

本测试的网络分簇 BWRAID 参数如下:使用 6 个 SN,每个创建 3 个 LV,总共 18 个;组成 6 个条带宽带为 3 的 RV,使 RV 均匀分布,即每个 SN 上各有 2 份数据 1 份校验;使用 6 个客户机,每个客户机分配 1 个 RV;随着客户机增加,更多的 SN 得到利用.

图 7 显示了客户机增加时的 IO 总带宽. 6 个客户机刚好充分利用 6 个 SN 的写带宽. 在客户机从 1 增加到 6 时,总的写带宽具有较好的线性增长. 3 个 RV 可充分利用 6 个 SN 的读带宽. 客户机从 1 增加到 3 时,总读带宽线性加速;当增加到 4～6 个客户端后,1 个磁盘同时服务 2 个顺序读,RAM 缓存对这种情况有较好的优化效果,总的读带宽没有大幅降低.

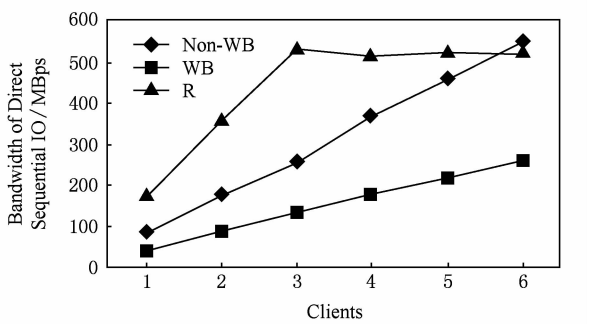


Fig. 7 IO bandwidth of network declustering BWRAID.

图 7 网络分簇 BWRAID 总带宽线

图 8 显示了 1 个 SN 故障后,降级时网络分簇 BWRAID 的 IO 总带宽. 在读命中 DCache 时,访问模式为 RAID1,因此总读带宽接近正常状态;当读没有命中 DCache 时(DCache read miss, R-Miss),

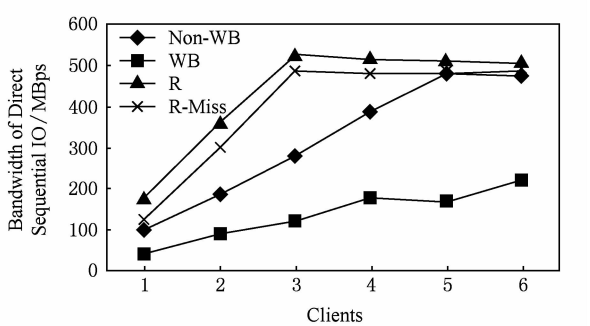


Fig. 8 IO bandwidth of degraded network declustering BWRAID.

图 8 网络分簇 BWRAID 降级带宽

读访问模式为降级 RAID4,RAM 缓存同时服务了降级读和本地读,总读带宽下降了约 40 MB. DCache 回刷前,写访问模式为 RAID1,因故障节点不提供服务,总写带宽相应减少;DCache 回刷后,写访问模式为 RAID4,降级 RV 进行“读改写”,总写带宽减少了约 61%.

4.5 测试扩展时间

本测试的网络分簇 BWRAID 参数如下:扩展前 3 个 SN,每个 SN 提供 4 个 32 GB 的 LV 构造 RV,总共 12 个 LV 和 4 个 RV. 使用在线迁移 LV 的方法[14],测试了系统扩增时间. 如图 9 所示,分别测试了 SN 数量从 3 个扩增到 4 个、从 4 个扩增到 5 个,以及从 3 个直接扩增到 5 个. 为了排除客户端读写影响,本测试没有客户机. 作为参照物,初始 BWRAID 中,每个 SN 分配 128 GB 的 LV.

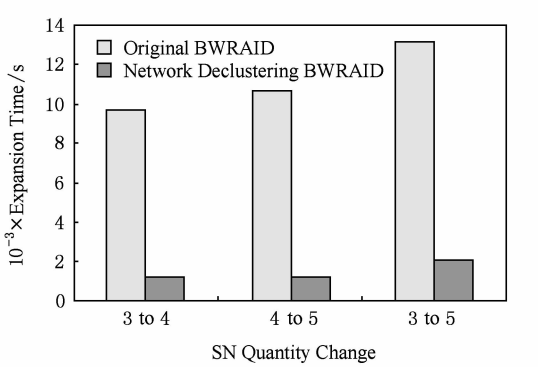


Fig. 9 Comparison of time consumption when expansion.

图 9 扩展时间对比

网络分簇 BWRAID 扩展效率提高了 5～8 倍. 扩展时,初始 BWRAID 改变了宽度,在迁移 LV 的同时还需要读取条带重新计算校验;而网络分簇 BWRAID 中 BWRAIDlet 构成没有变化,仅迁移了 LV. 另一因素是迁移数据量不同,为了维持数据块级别的全对称结构,初始 BWRAID 要迁移大部分的数据,本例中迁移了总数据量的 70%～80%;而网络分簇 BWRAID 使用本文的扩展算法迁移最少的 LV,在本例中增加 1 个节点迁移了 64 GB,增加 2 个节点迁移 128 GB,为总数据量的 20%～40%.

4.6 测试并行恢复时间

下面给出网络分簇 BWRAID 的恢复性能. 为了避免客户端读写影响测试结果,本测试没有客户机. 本测试的网络分簇 BWRAID 参数如下:12 个 SN,RV 宽度为 3,LV 大小为 32 GB,均匀分布. 分别测试了不同数量 LV 时的恢复时间.

图 10 给出了网络分簇 BWRAID 并行恢复的时

间,恢复加速比跟并行数量呈线性增长.本文算法选择在不相交叠的 RV 上进行并行恢复,这样 2 个并行的恢复进程没有竞争资源,可最大化并行恢复的加速比.在 12 个 SN 上(包括故障节点),算法可选择 4 个 RV 并行恢复.网络分簇 BWRAID 的顺序恢复时间也比初始 BWRAID 的时间少,因为 DDP 布局更简单,每次恢复 1 个 LV 数据或校验;而初始 BWRAID 全对称,1 个 SN 需要同时恢复数据和校验,恢复数据流复杂.

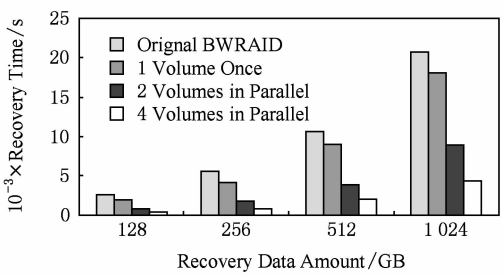


Fig. 10 Time consumption of parallel recovery.

图 10 并行恢复时间

5 相关研究

MiPiL[15]在数据块级别实现均衡,扩展时维持统一的数据布局,能够移动最少的数据,仍然需要计算校验.另外,MiPiL 的研究中没有涉及并行迁移或恢复.

PBM[16]按数据块实现均衡,维持固定模式的布局,在扩展时能够移动最少的数据,也不需要重新计算校验.然而,为了实现多次扩展,PBM 要求双倍扩展,即新添加的 SN 数目等于原有 SN 数目.同样,PBM 的研究中也没有涉及并行迁移或恢复.

DiskReduce[5]在 Hadoop 的 HDFS 基础上实现副本转"目录内 RAID"存储,数据块均衡.迁移或恢复转换为 MapReduce 的工作任务,而 MapReduce 任务可并行执行.文献[5]中并没有给出并论证具体的扩展或恢复算法.

初始 BWRAID 采用按数据块的均衡,使用映射公式,将多个 SN 按全对称的结构组织.扩展需要迁移大部分的数据,并重新计算校验.恢复时,数据集中恢复到单一节点.

本文的网络分簇 BWRAID 以容量相等的小逻辑卷为单位管理存储空间,在逻辑卷级别实现均衡,使用随机分配算法,除了 BWRAIDlet 的 DDP 布局,没有其他统一或固定的数据布局.本文的扩展算法移动最少的逻辑卷,不需要计算任何校验,能并行迁移.本文的并行恢复算法能够利用多 SN 成倍加速恢复.

6 结论及后续工作

本文给出了网络分簇 BWRAID 使用逻辑卷组成的非对称结构及方法,并测试了相对于初始 BWRAID 的改进,包括 3 方面内容:1)网络分簇 RAID 降低扩展开销,提高扩展速度.给出并分析了搜索目标逻辑卷的算法.结合在线迁移逻辑卷的技术,系统扩展时不需要计算校验,同时服务用户的读写请求.2)网络分簇 RAID 多卷并行恢复,成倍加快了数据恢复速度.3)采用 DDP 布局优化 BWRAID 的数据布局,提高了性能,并简化了设备堆栈.

BWRAID 的后续研究工作主要是实现多节点冗余保护.本文分簇 BWRAID 为单冗余,扩展到多冗余需要维持多冗余一致性的方法.例如,RAID6 控制器所在节点故障会打断更新操作引入不一致,而解决该问题的常规方法"重新同步"不能在降级时使用.因此,有必要研究多冗余的网络 RAID 的一致性方法.

参 考 文 献

[1] Chen P M, Lee E K, Gibsion G A, et al. RAID: High performance, reliable secondary storage [J]. ACM Computing Surveys, 1994, 26(2): 145-185

[2] Na Wenwu. Research on the network RAID and its key technologies [D]. Beijing: Institute of Computing Technology, Chinese Academy of Sciences, 2009 (in Chinese)
(那文武. 网络 RAID 及其关键技术研究[D]. 北京:中国科学院计算技术研究所, 2009)

[3] Na Wenwu, Ke Jian, Zhu Xudong, et al. BW-netRAID: A network RAID system with backend centralized redundancy management [J]. Chinese Journal of Computers, 2011, 34(5): 912-923 (in Chinese)
(那文武, 柯剑, 朱旭东, 等. BW-netRAID:一种后端集中冗余管理的网络 RAID 系统[J]. 计算机学报, 2011, 34(5): 912-923)

[4] Wilkes J, Golding R A, Staelin C, et al. The HP AutoRAID hierarchical storage system [J]. ACM Trans on Computer Systems, 1996, 14(1): 108-136

[5] Fan B, Tantisiriroj W, Lin Xiao, et al. DiskReduce: RAID for data-intensive scalable computing [C] //Proc of the 4th Annual Workshop on Petascale Data Storage. New York: ACM, 2009: 6-10

[6] Holland M, Gibson G A. Parity declustering for continuous operation in redundant disk arrays [C]//Proc of the 5th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 1992: 23-35

[7] Holland M, Gibson G A, Siewiorek D P. Architectures and algorithms for on-line failure recovery in redundant disk arrays [J]. Distributed and Parallel Databases, 1994, 2(3): 295-335

[8] Ma Yili, Fu Xianglin, Han Xiaoming, et al. The separation between storage and computation [J]. Journal of Computer Research and Development, 2005, 42(3): 520-530 (in Chinese)
(马一力, 傅湘林, 韩晓明, 等. 存储与计算的分离[J]. 计算机研究与发展, 2005, 42(3): 520-530)

[9] Calder B, Wang Ju, Ogus A, et al. Windows azure storage: A highly available cloud storage service with strong consistency [C] //Proc of the 23rd ACM Symp on Operating Systems Principles. New York: ACM, 2011: 143-157

[10] Rosenblum M, Ousterhout J K. The design and implementation of a log-structured file system [J]. ACM Trans on Computer Systems, 1992, 10(1): 26-52

[11] Meng Xiaoxuan, Si Chengxiang, Na Wenwu, et al. P-Cache: Providing prioritized caching service for storage system [C] //Proc of the 7th IEEE Int Symp on Parallel and Distributed Processing with Applications. Piscataway, NJ: IEEE, 2009: 3-10

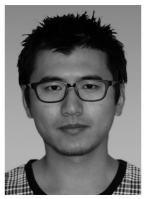
[12] Mellanox Technologies Ltd. InfiniBand [EB/OL]. [2014-12-01]. http://www.mellanox.com/pdf/whitepapers/WP_2007_IB_Software_and_Protocols.pdf

[13] Bolkhovitin V. iSCSI-SCST [EB/OL]. [2014-12-01]. http://scst.sourceforge.net

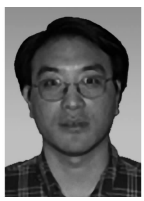
[14] Liu Chang. Research and implementation of data migration framework in BW-RAID system [D]. Beijing: Institute of Computing Technology, Chinese Academy of Sciences, 2012 (in Chinese)
(刘昌. BW-RAID系统中数据迁移框架研究与实现[D]. 北京: 中国科学院计算技术研究所, 2012)

[15] Zhang Guangyan, Zheng Weimin, Li Keqin. Rethinking RAID-5 data layout for better scalability [J]. IEEE Trans on Computers, 2014, 66(11): 2816-2828

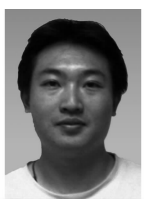
[16] Mao Yu, Wan Jiguang, Zhu Yifeng, et al. A new parity-based migration method to expand RAID-5 [J]. IEEE Trans on Parallel and Distributed Systems, 2014, 25(8): 1945-1954



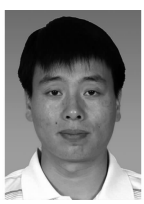
Sun Zhenyuan, born in 1983. PhD. Student member of China Computer Federation. His main research interests include network storage and high availability.



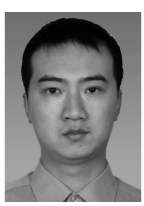
Xu Lu, born in 1962. PhD, professor and PhD supervisor. His main research interests include cluster file systems and virtual storage (xulu@bwstor.com.cn).



Liu Zhenjun, born in 1976. PhD, associate professor and master supervisor. His main research interests include network storage and virtual virtualization (liuzhenjun@nrchpc.ac.cn).



Dong Huanqing, born in 1976. PhD and software engineer. His main research interests include software engineering and virtual storage (donghuanqing@bwstor.com.cn).



Liu Chang, born in 1985. Master and software developer. His main research interests include network storage (liuchang@bwstor.com.cn).